ESL-TR-02/02-01

Automated Data Quality Analysis Research and Development

LoanSTAR Deliverable Report

Final Report

James Sweeney, Jr.
Jeff S. Haberl, Ph.D., P.E.

February 2002

PREFACE

This report is one of a series of reports that documents the development of the LoanSTAR and Technical Assistance deliverables. The developments are broken down into two divisions, Task C and Task D. The following two tables itemize the deliverables for each Task. The information included in this report represents the LoanSTAR and Technical Assistance deliverable Task D-Number Four.

Task C Deliverables	Description
1.Access and Display LoanSTAR Data via the Internet for Interested Agencies	Energy Systems Lab's web-based interface to the LoanSTAR Energy Databases, referred to as the Energy and Environmental Data System (EEDS).
2.Report Print/ Viewing Options (i.e., PDF format)	Effort to move reporting to Adobe Inc.'s Portable Document Format.
3.Online Feedback Ability (automated email to ESL)	Online feedback form built into the ESL's EEDS.
4.Internet Data Logging and Display a.Develop Internet-Based Real-time Display Software b.Buy/Develop and Test a Date Logger with Integrated TCP/IP Connectivity c.Buy/ Develop and Test Software to Poll a TCP/IP Logger	Research and development of internet based power measurement and energy data logging techniques.

Task D Deliverables	Description
1.Automated Email Polled Data Problem Alerts. (ESL internal)	The automatic notification via email of the sites that failed polling for a given polling job.
2.Automate IPNs Review Process to an 'Exception Only' Basis (ESL internal)	Streamlining the IPN review process by improving problem tracking and reporting.
3.Convert Ipns plots to PDF format (ESL internal)	Effort to move IPN reporting to Adobe Inc.'s Portable Document Format.
4.Automated the Data Analysis/Quality Verification Process to 'Exception Only' (ESL internal)	A series of checks that monitor raw files loaded into the database for low level data quality analysis.
5.Move IPNs, MECRs, AECRs to an Internet Base	Effort to combine reporting in Adobe Inc.'s Portable Document Format with a secure Web-based interface.

EXECUTIVE SUMMARY

The information included in this report represents the LoanSTAR and Technical Assistance deliverable Task D, number four, "Automated the Data Analysis/Quality Verification Process to 'Exception Only' (ESL internal)". This work reports on the Automated Data Quality process development. This process is a three-phase development of a series of checks that monitor polling information and raw files loaded into the database for low to high level data quality analysis. Phase one and two are low-level polling and data quality checks and are the subject of this report.

ACKNOWLEDGEMENTS

This project would not have been possible without the support that was provided by the following persons and/or the agencies or companies for which they work: Ms. Theresa Sifuentes and Mr. Dub Taylor at the Texas State Energy Conservation Office,

This project also would not have been possible without the support that was provided by the following persons that work at the Texas A&M University's Energy Systems Lab: Dr. Charles Culp, and Yoon-Cheung Chang.

TABLE OF CONTENTS

PREFACE	1
EXECUTIVE SUMMARY	2
ACKNOWLEDGEMENTS	3
1.0 INTRODUCTION	5
2.0 DESIGN DOCUMENTATION	5
3.0 IMPLEMENTATION DOCUMENTATION	12
4.0 FUTURE DIRECTIONS	17
5.0 SUMMARY	17
6.0 CODE	18

1.0 INTRODUCTION

The Automated Data Quality process development is an effort to automate the analysis and monitoring of the Energy Systems Lab's (ESL) polling and energy data process. Polling status refers to basic telecommunication line states, quality of data communications, and logger-system checks. The data from the Data Logging System can also be checked for quality at various stages in the ESL's Data System.

2.0 DESIGN DOCUMENTATION

2.1 ESL' Data System Overview

The ESL's Data System is a set of technological tools that acquire, organize, and process energy data. Specifically, the Data System is comprised of relational databases and database applications that retrieve and manipulate sets of energy data. These include in-house derived applications that retrieve the data from the remote loggers, load the data into the database, perform savings calculations and help manage the configuration of the system (see Figure 1). Prior to this effort, data quality of the system was largely derived through a manual, graphic review process. This process, although accurate, is cumbersome in its scope and implementation. Automation of the "lower-level" data quality checking will enabled a more refined review process.

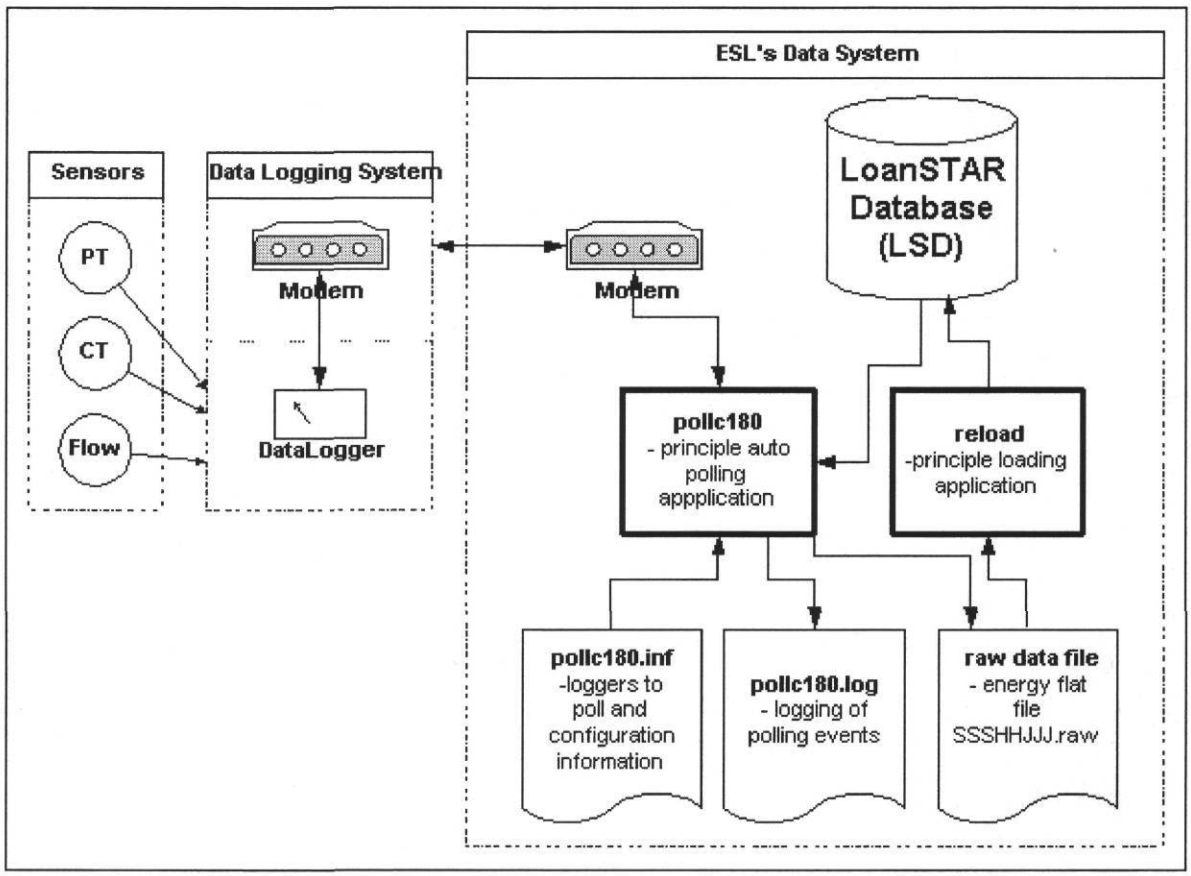

Figure 1. Energy Core Polling and Reloading Process Diagram

2.2 Definitions and Process Descriptions

Data quality is composed of different tiers of information throughout the system. These tiers can be defined by their position in the data processing stream. There are three major tiers in this stream: 1) the telecommunications and download tier, 2) the load-time tier and 3) the offline tier (see Figure 2).

Figure 1 displays the major data acquisition and loading portions of the ESL's Data System and remote monitoring interfaces. The polling and loading process are critical areas where data quality information may be acquired.

2.3 Telecommunications and Download Tier

The telecommunications and download tier is the first tier in the data acquisition and loading stream (Figure 2). This tier of data quality is concerned with the quality and recording of the telecommunication activity. This includes basic telecommunication line states (busy, disconnected) and the quality of the data communications over the telecom link. The download data quality includes information on the logger/data system synchronization checks (time and parameters), and other downloading quality checks.

2.3.1 Process Functions and Input/Output

Functions:

- Check polling status (OK or FAILED) from pollc180 queue output (pollc180 –q).
- Check telecommunication state (busy, disconnected, connection established).
- Check telecommunication data quality (analog check, read verification, cyclical redundancy checks, etc…).
- Store results to log files.
- Analyze log files and post results in "appropriate" data quality tables for review.

Inputs :

- Pollc180 -q : Yields output with the sites and serial numbers that failed for the week.
- Pollc180.log : File that contains telecom state, telecom data quality information for each polling.
- Manual polling updates (spreadsheet or direct web form updates).

Outputs:

- Process pollc180 -q and pollc180.log by joining to form faillist with the following data:
  - Julian week
  - Site number
  - Serial number
  - Error description
  - Download type (manual/auto)

- pollc180.log database load
- Manual polling updates to database

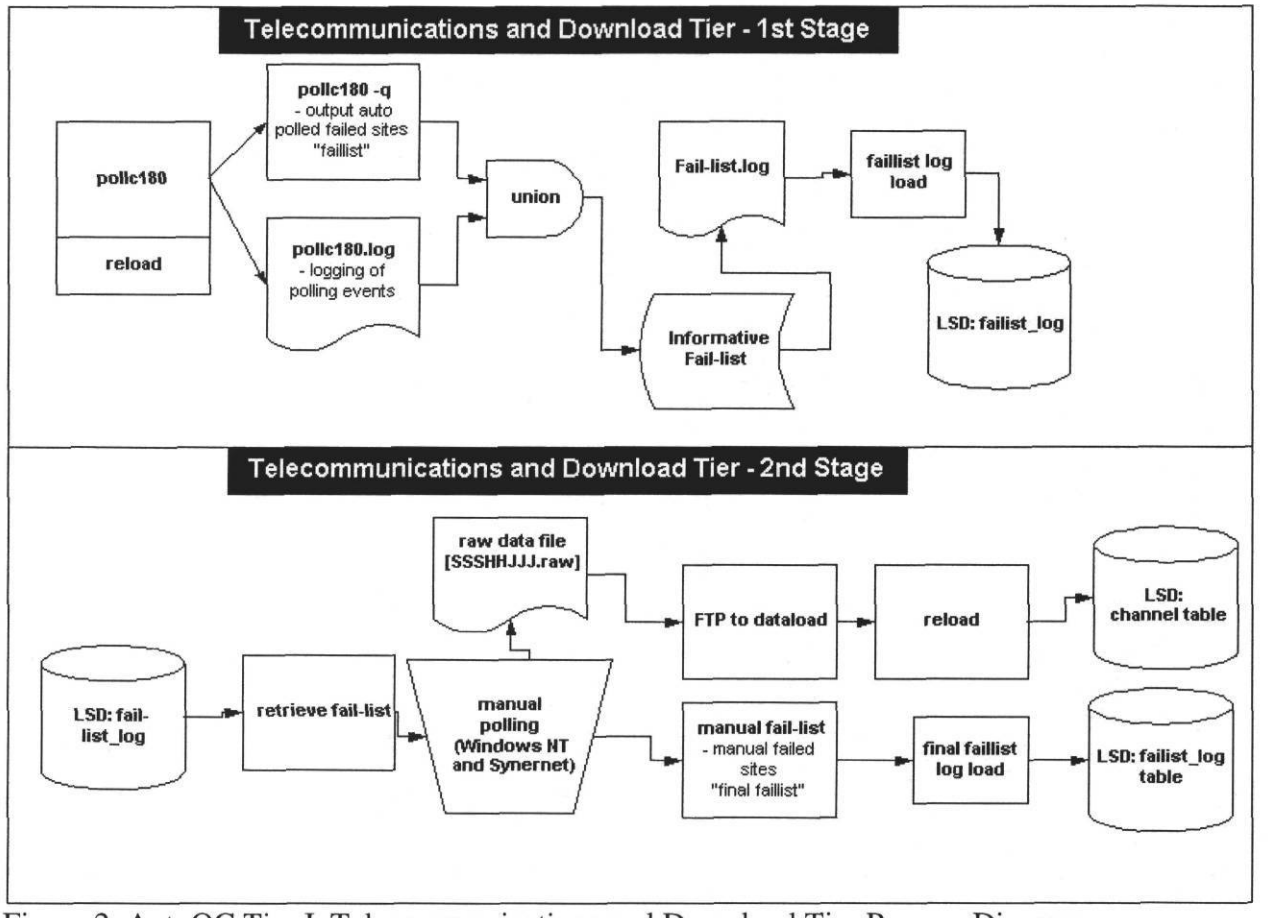

Figure 2. AutoQC Tier I. Telecommunications and Download Tier Process Diagram.

2.4 Load-time Tier

The load-time tier is a system of quality checks and algorithms that search for data quality when the energy data are loaded into the relational database. Simple non-computationally intense operations occur at this tier. Mining is divided into two distinct modes of operation, "blast" and "bore". Blast being load-time analysis only and bore mining being data quality analysis that incorporates historical or specific non-load-time information. Tier II is blast mining only (see Figure 3).

2.4.1 Classification of Blast Mining Functions

The following are descriptions of the different functions that may be provided by this tier of data quality. This process is intended to observe obvious trends in data. This processing will occur relatively fast at the load-time tier. Blast mining checks for missing records (time series inconsistencies), zeros, and –99s (missing data in ESL's terminology). Blast mining also scans the Power fail and out of range tables to check for other indicators of a data channels integrity.

Missing record (time series inconsistencies):
If not continuous data set load to missing_record table.
Table structure:
ch####, begin datetime stamp, end datetime stamp

Zero:
Check check_zeros and out_of_range table.
Table structure:
ch####, datetime stamp, value (0.0 or some limit to it)

-99:
Scan for -99 in the existing out_of_range table.

Power fail:
The reload program scans for this and stores the results in the existing power_fail table.

Data communication states:
Scan for certain logger dependent data communication states.

Out of range and negative:
Store high and low checks in the out_of_range table.

2.4.2 Process Functions and Input/Output

The blast data miner process consists of inputs, outputs and functions that process the data and provide output for reviewers. Inputs to this process are the individual raw files and the data quality tables. The output of the process is data quality information presented to the reviewer via the data quality tables (see Figure 3). The blast data miner module is attached to the reload application. This facilitates the processing via these functions at the load-time. Figure 3 shows the overall process and its location in the data loading process. The following is a list of the functions, inputs and outputs of the blast data mining operation.

Functions:
- Missing Records (no TSRs)
- Missing data (TSR is present but -99 or zero)
  - Zero
  - -99 (from the datalogger)
- Power fail or logger dependent data transmission quality checks
- Out of range checks
  - High and low limit check of raw data versus the pre-determined fixed limits in the database
  - Load out_of_range table and not the channel table

Inputs:
- Individual raw file to load
- Data quality database tables

Outputs:

- Results to appropriate database tables

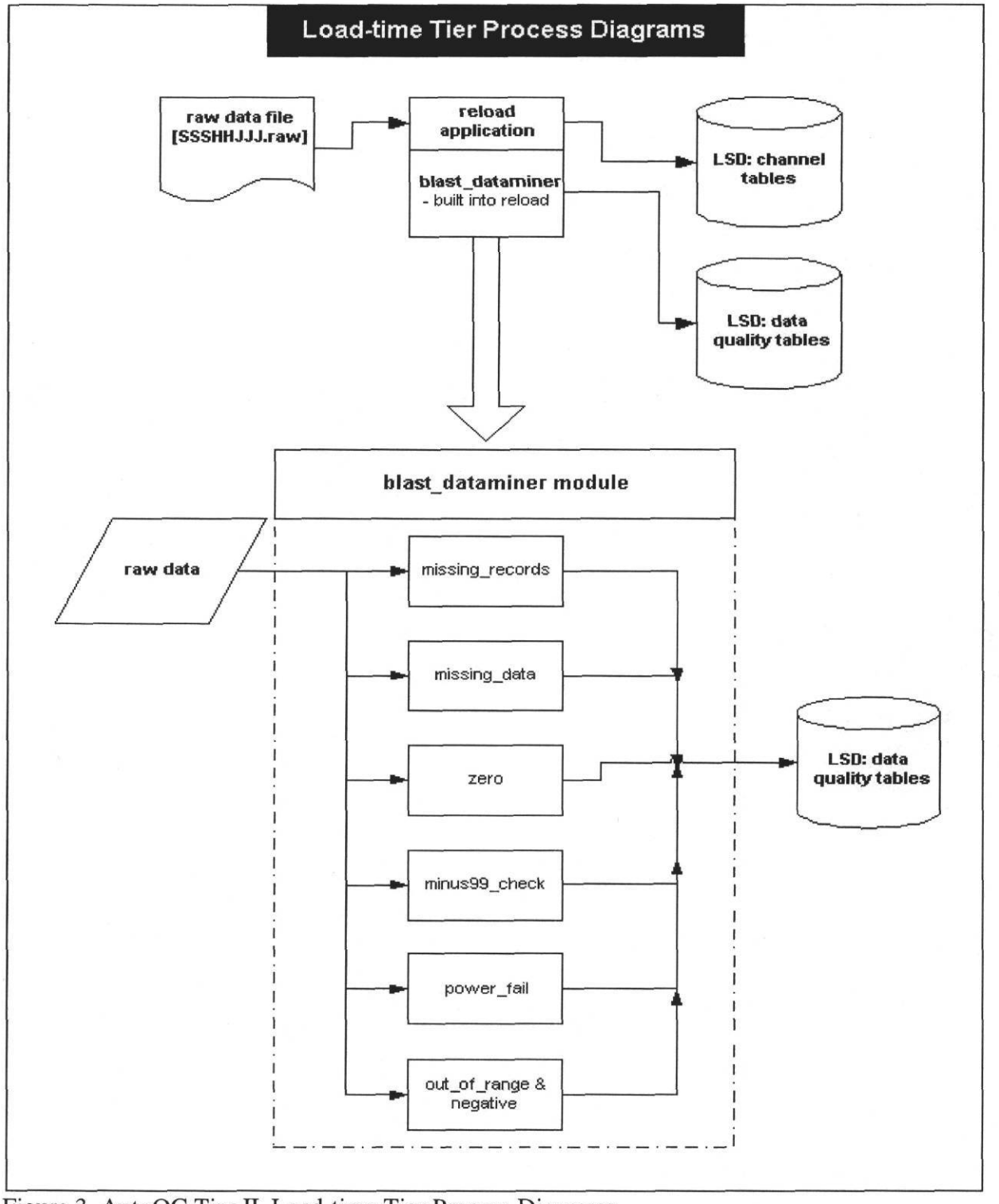

Figure 3. AutoQC Tier II. Load-time Tier Process Diagrams.

2.5 Offline Tier

The third tier is the offline tier, or the data quality check that is built offline of the data acquisition and loading processes. Complex computationally intense data mining operations and construction occur at this tier. This type of data quality mining is described as "Bore" mining to indicate more granularity in the mining operation.

2.5.1 Bore mining features

Bore mining features are more complicated in nature than its blast mining counterpart. In this Tier of analysis statistics and predictive channel typing are utilized. Some basic features of this type of data quality analysis are:

- Channel Typing - Channels are analyzed and evaluated based on the type of channel it may be, i.e. electrical analog, electrical digital, outside air temperature, etc…
- Behavioral and predictive analysis - Channels are analyzed based on historical trends.
- Knowledge based -Channels are analyzed based on specific knowledge about the channel.

2.5.2 Basic Approach

Direct database analysis is achieved by utilizing the existing getdatc routine to retrieve data then piping the results retrieved into the analysis function. Benefits of this approach are potential speed of development and flexibility. The previously developed Getdatc application contains the ability to retrieve channel data over discrete time frames. Another approach would be to do an embedded sql routine to perform the analysis functions.

File system analysis can be performed much faster but does not directly reflect what is in use (i.e. it is raw, unprocessed data). After data are loaded it is often nulled and rescaled for various reasons.

This tier will be implemented in future research and development efforts.

3.0 IMPLEMENTATION DOCUMENTATION

3.1 Automatic Data Quality Process -Tier 1

The principal objective of Tier 1 of the Automatic Data Quality process is the monitoring of the telecommunication and polling data logs. This process loads logging information for sites that failed in a given week which were polled by the Energy Systems Lab. This process uses various C and Perl programs to process the outputs of other upstream programs to consolidate telecommunication and polling problems for the week. The output of this process is written to an ASCII flat file and is loaded into the ESL's databases.

Inputs:

- */prod/raw/pollc180.log*:

A flat file that contains logging information of sites polled for the week. This file contains information for multiple weeks as well as that for the week. For each week, site number, baud rate, phone number, and error descriptions are listed in this file.

- *pollc180 –q > poll.txt*:

Lists sites and associated serial numbers that failed in the polling for the week.

Output:

- *faillist.mmmddyy*:

The Polling Fail-List for the week with the following data: site number, serial number, baud rate, phone number, and error descriptions. This data is loaded into the database by the faillist_load program.

Key Modules:

- *create_logfiles( )*:

This process makes intermediate results. The resulting files are used as inputs to *union( )*. To deal with inputs beyond the stack capacity, *pollc180.log* with information for multiple weeks as well as that for the week are split into sub-files with data for only one week. The results are the sub-files and *polling_list.txt* file. The pollig_list.txt file lists the names of the sub-files created.

- *union( )*:

pollc180.log combines with the *poll.txt* to get the data processed for the week in process. The final result of *faillist.mmmddyy* is generated from this process.

- *faillist_load( )*:

Loads *faillist.mmmddyy* generated weekly to the *faillist* table in the database (LSD/ESLD).

Additional Info:

- Tier 1 is composed of three major modules: 1) *create_logfiles( )*, 2) *union( )* and 3) *faillist_load( )*. Since the original input of *pollc180.log* contains many redundancies, *Perl scripts* are used to filter out redundancies and are executed during runtime.

Figure 4 displays a block diagram of the Automated Data Quality Process Tier 1. It shows how the various inputs and outputs are processed and loaded into the 'faillist' table in the database. This process begins by combining the polling logs with the current week of polling problems. The polling logs are processed into week chunks (if necessary) and are combined with the current polling queue. The polling queue are those loggers which remain to be polled. These two sources produce an informative 'faillist' for the week. The data contained in this file is then loaded into the database.

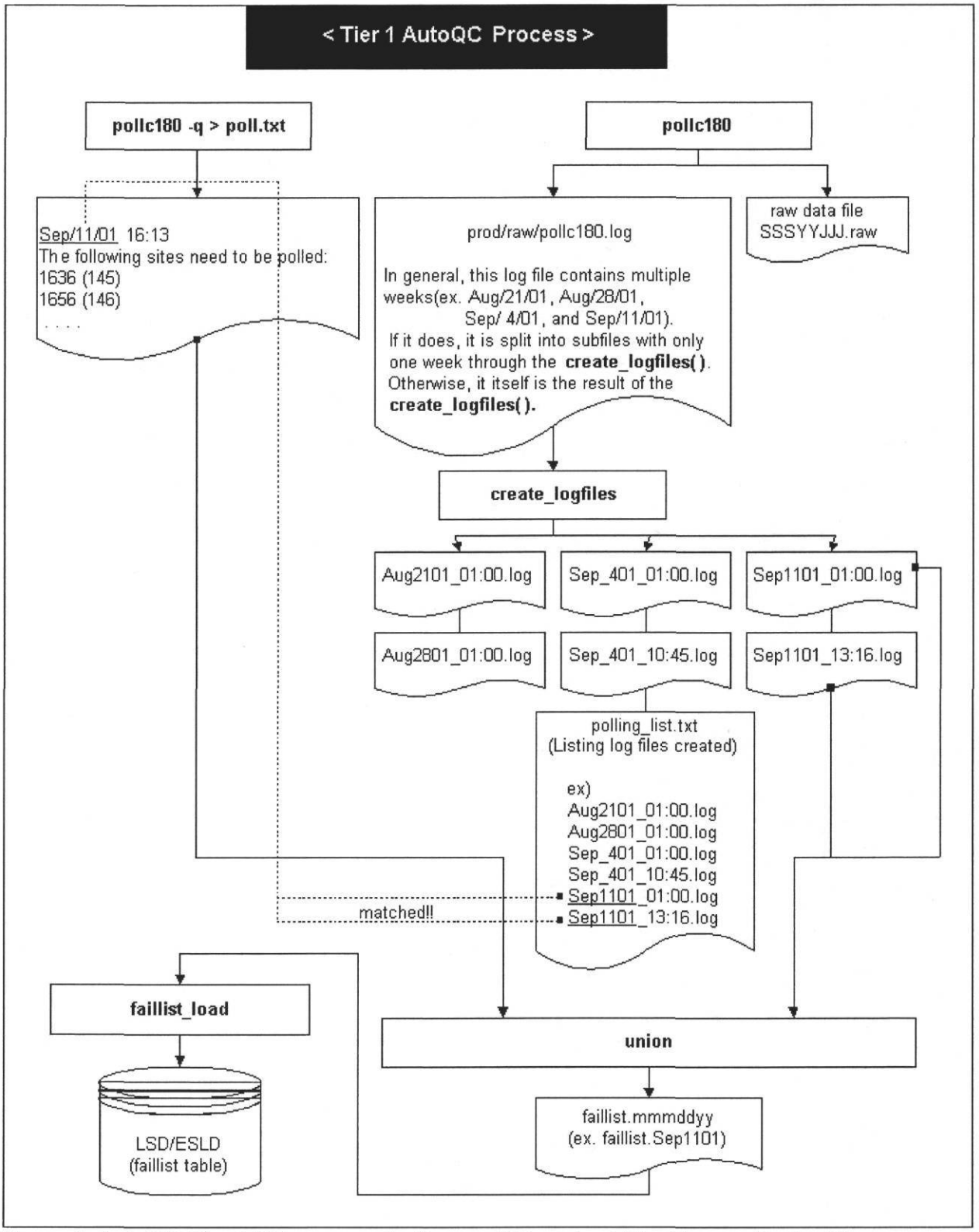

Figure 4. AutoQC Tier I. Telecommunications and Download Tier Process Design Diagram.

3.2 Automatic Data Quality Process -Tier 2 - Blast Mining

The principal objective of Tier 2 of the Automatic Data Quality process is the analysis of the raw energy files for "low-level" data quality indicators. This process analyzes the raw files at load-time, or when the raw files are loaded into the database. The results are stored in the database for viewing by the data quality staff. This process uses various C and Perl programs and is attached to the core loading application 'reload'. When data is loaded into the database it is analyzed for "low-level" quality indicators.

"Low-level" quality checks are the power status of the logger, out of range checking, zero value and missing record (gaps in the time series) tracking. The output of this process is written to a flat file and is loaded into the ESL's databases.

Input:
- output of the reload process – SSSYYJJJ.raw_loaded

Mining Functions:
- Power Fail information : Flags (F/U/O/N) in the raw file
- Values out of range (not within the site specific high and low limits)
- Zero values
- Missing records (gaps in the time series)

Intermediary results:
- power_fail_log.log
- out_of_range.log
- zeros.log
- missing_records.log

These files are removed after the final output file is generated.

Output:
- SSSYYJJJ_autoqc2
- Update the following tables.

  Tables:
  - power_fail_log
  - out_of_range
  - zeros
  - missing_records

Figure 5 displays a block diagram of the Automated Data Quality Process Tier 2. It shows how the various inputs and outputs are processed and loaded into the channel and data quality tables in the database. This process consists of generating log files for the principal data mining functions. All of the data in these log files is loaded into their respective tables in the database. All the log file data is also combined into a logger specific text file (See Figure 5).

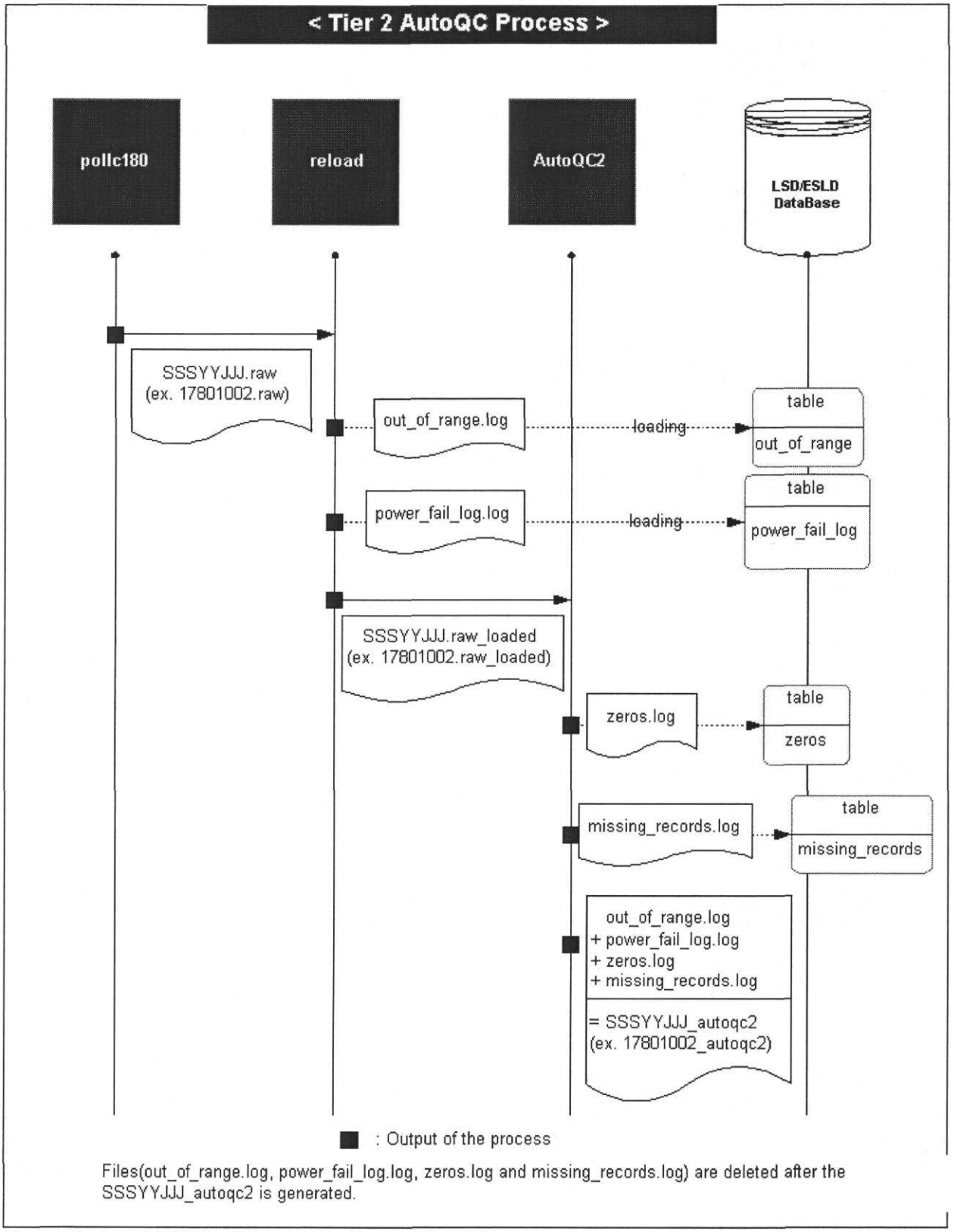

Figure 5. AutoQC Tier II. Data Quality Design Diagram.

4.0 FUTURE DIRECTIONS

In the future the ESL will begin the research and development of the third tier of the Automated Data Quality process which is the offline tier. These data quality checks will be built offline of the data acquisition and loading processes. Complex computationally intense data mining operations and construction occur at this tier. This type of data quality mining is described as "Bore" mining to indicate more granularity in the mining operation. In this Tier of analysis statistics and predictive channel typing are utilized. Some basic features of this type of data quality analysis are more 'knowledge-based' where by data channels are analyzed according to type and their behavior is analyzed accordingly. The general approach for this tier is achieved by utilizing the existing getdatc routine to retrieve data then piping the results retrieved into the analysis functions.

5.0 SUMMARY

The information included in this report documents the LoanSTAR and Technical Assistance deliverable Task D, number four, "Automated the Data Analysis/Quality Verification Process to 'Exception Only' (ESL internal)". This process is a three-phase development of a series of checks that monitor polling information and raw files loaded into the database for low to high level data quality analysis. Prior to this effort, data quality of the system was largely derived through a manual, graphic review process. This process, although accurate, is cumbersome in its scope and implementation. Automation of the "lower-level" data quality checking will enabled a more refined review process with the implementation of Tier 1 and II processes as outlined herein. Tier I of data quality is concerned with the polling results, recording of the telecommunication activity and storing this information an organized manner in the database. This process loads logging information for sites that failed in a given week which were polled by the Energy Systems Lab. This process uses various programs to process the outputs of other upstream programs to consolidate telecommunication and polling problems for the week. Tier II data quality work is the energy data quality portion. This is a system of quality checks and algorithms that search for energy data quality when the energy data are loaded into the relational database. This represents the blast mining functions.

6.0 SYSTEM CODE

6.1 Automatic Data Quality Process-Tier 1 (Source)

filter_str

```
||=========================================================
||  This perl script is used in the faillist.ec.
||  Usage      : to remove redundancies of pollc180.log
||  Copyright  2001-2002,
||  Texas Engineering Experiment Station,
||  Texas A&M University.
||  Architect: Jim Sweeney, Programmer: YoonCheung Chang
||=========================================================


#!/usr/local/bin/perl -w
# Usage : filter_str
# remove line with 'str'

while (<>) {
    if (/Successful exit./) {
    }
    else {
        s/time [0-9]+, //g;

       s/(read end tsr failed!, error shutdown.write to serial port failed, )+/read end tsr
failed!, error shutdown./g;

       s/(write to serial port failed, CRC error,)+//g;
       s/(write to serial port failed,CRC error, )+//g;
       s/(write to serial port failed, CRC error, )+//g;
       s/(write to serial port failed,CRC error,)+//g;
       s/(write to serial port failed, CRC error, write to serial port failed,CRC error,)+//g;

       s/(write to serial port failed,  read start tsr failed!, error shutdown. )+//g;
       s/(write to serial port failed,  read start tsr failed!, error shutdown.)+//g;
       s/(write to serial port failed, read start tsr failed!, error shutdown. )+/read start tsr
failed!, error shutdown./g;
       s/(write to serial port failed, read start tsr failed!, error shutdown.)+/read start tsr
failed!, error shutdown./g;

       s/(read start tsr failed!, error shutdown. read start tsr failed!, error shutdown.)+//g;
       s/read start tsr failed!, error shutdown. read start tsr failed!, error shutdown.//g;

       s/read start tsr failed!, error shutdown.read start tsr failed!, error shutdown./read
start tsr failed!, error shutdown. /g;
       s/read start tsr failed!, error shutdown. read start tsr failed!, error shutdown. /read
start tsr failed!, error shutdown. /g;
       s/read start tsr failed!, error shutdown. read start tsr failed!, error shutdown. /read
start tsr failed!, error shutdown. /g;

       s/(CRC error, read start tsr failed!, error shutdown. read start tsr failed!, error
shutdown. )+/CRC error, read start tsr failed!, error shutdown. /g;
       print;
    }
}
```

delete_bracket

```
||=========================================================
||  This perl script is used once in the faillist.ec.
||  Usage      : to delete brackets
||  Copyright  2001-2002,
||  Texas Engineering Experiment Station,
||  Texas A&M University.
||  Architect: Jim Sweeney, Programmer: YoonCheung Chang
||=========================================================


#!/usr/local/bin/perl -w
# Usage : delete_bracket
# delete only brackets
```

```
while (<>) {
        if (/[a-zA-Z]/) {
        }
        else {
        s/[()]//g;
        print;
        }
}
```

create_log.c

```
||============================================================
||  usage : to extract only data for the week from pollc180.log
||  input : /prod/raw/pollc180.log
||  output: /usr/local/lsd/src/autoqc/created_log
||          this file includes the data extracted.
||          this will be used, and then deleted by faillist.ec
||  Copyright  2001-2002,
||  Texas Engineering Experiment Station,
||  Texas A&M University.
||  Architect: Jim Sweeney, Programmer: YoonCheung Chang
||============================================================


#include <string.h>
#include <stdio.h>
#include <stdlib.h>

FILE *r_file, *r_pollc;
FILE *w_pollc;

int main(void)
{

    int  ch, index, pollc_ch, pollc_index, new_pollc_ch;
   char  buffer[20] = {0},  pollc_buffer[20] = {0};
   long  position;

   /*-------------------------------------------------------*/
   /*--------- CATCH THE CURRNET LOGGING DATE --------------*/
   /*-------------------------------------------------------*/

   /* system("pollc180  -q > /usr/local/lsd/src/autoqc/poll.txt");*/

   if ((r_file = fopen ("/usr/local/lsd/src/autoqc/poll.txt","r")) == NULL)
         printf("The file not opened\n");

   ch = fgetc(r_file);

   while(ch == 10)
              ch = fgetc(r_file);

   index=0;
   while(ch != 32) {
              buffer[index++]=ch;
              ch = fgetc(r_file);
   }
   buffer[index]='\0';
   printf("current logging date: %s\n",buffer);

   /*-------------------------------------------------------------*/
   /*------ FIND THE DATE MATCHED WITH CURRENT LOGGING DATE -----*/
   /*-------------------------------------------------------------*/

   if ((r_pollc = fopen ("/prod/raw/pollc180.log","r")) == NULL)
         printf("The file not opened\n");

   pollc_ch = fgetc(r_pollc);

   while(pollc_ch == 10)
              pollc_ch = fgetc(r_pollc);

   position = ftell(r_pollc);

   pollc_index=0;
   while(pollc_ch != 32) {
               pollc_buffer[pollc_index++]=pollc_ch;
```

```
            pollc_ch = fgetc(r_pollc);
    }
    pollc_buffer[pollc_index]='\0';
    printf("Initial date: %s\n",pollc_buffer);

    while(strcmp(buffer, pollc_buffer) != 0)
    {
       while(pollc_ch != 10)
                pollc_ch = fgetc(r_pollc);

       pollc_ch = fgetc(r_pollc); /* the first character of the line*/

       position = ftell(r_pollc);
       position--;  /* file positon of the 1st character for the line */

       pollc_index=0;
       while(pollc_ch != 32) {
                pollc_buffer[pollc_index++]=pollc_ch;
                pollc_ch = fgetc(r_pollc);
       }
       pollc_buffer[pollc_index]='\0';
    }

    printf("Matched date: %s\n",pollc_buffer);

    /*setting the file pointer to the position location */
    if (fsetpos(r_pollc, &position) != 0)
        printf("fsetpos error\n");

    /*----------------------------------------------------------------*/
    /*-- MAKE THE FILE INCLUDING THE ONLY CURRENT LOGGING DATE INFO --*/
    /*----------------------------------------------------------------*/

    if ((w_pollc = fopen("created_log","w")) == NULL)
           printf(" The file not existed\n");

    new_pollc_ch = fgetc(r_pollc);
    while(new_pollc_ch != EOF){
          fputc(new_pollc_ch, w_pollc);
          new_pollc_ch = fgetc(r_pollc);
    }

    fclose(r_file);
    fclose(r_pollc);
    fclose(w_pollc);

    return 0;
}
```

faillist.ec

```
||=====================================================================
||  usage : to get logging information for sites to be polled again
||          from created_log and to load it on the table lsd@lsol:faillist
||  logging information loaded :
||  date,site_number,serial_number,baud_rate,phone_number,error_description
||  input : /usr/local/lsd/src/autoqc/created_log
||  output: /usr/local/lsd/src/autoqc/faillist.sssyyjjj_loaded
||          this includes logging information to be loaded on the table
||  Copyright  2001-2002,
||  Texas Engineering Experiment Station,
||  Texas A&M University.
||  Architect: Jim Sweeney, Programmer: YoonCheung Chang
||=====================================================


#include<string.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

$include sqlca;
$include sqltypes;

#define MAX_LINE 100
#define MAX_NUM_FILES 10
#define MAX_CHAR 10
```

```
 int Calculate_Fragments(char buffer[MAX_LINE]);
void Split_into_Fragments(FILE *r_buffer, FILE *w_buffer);
void faillist_load(char *, char *);
void insert_rows(char *, int, int, char *, char *, char *);
 int convert_to_integer(char *);

FILE *read_info,*read_log,*r_buffer,*r_dates;
FILE *w_stream, *w_buffer;
FILE *tmp_ptr;

int main(void)
{

  char line[MAX_LINE], date[MAX_LINE],time[MAX_LINE],temp[MAX_LINE];
  char *name,buffer[MAX_LINE],command[MAX_LINE];
  char dates[20]={0}, cur_dates[50]={0}, convert[5]={0}, month[5]={0};
   int mo, dy, yr, hr, mn;
   int i,index,ch;
   int num_fragments;
  char *mid[MAX_NUM_FILES];
  char num[10];
   int j, n, c, k, m;

  /*********************************************/
  /******** MAKE THE TABLE TIME FORMAT ***********/
  /*********************************************/

  if ( (r_dates = fopen ("/usr/local/lsd/src/autoqc/poll.txt","r")) == NULL)
        printf("The file not opened\n");

  index =0;
  ch = fgetc(r_dates);
  while(ch == 10)
        ch = fgetc(r_dates);

  while(ch != 10) {
        if (ch == 95)/* ASCII code 95 is matched with the '_' key*/
            dates[index++]= 32;
        else
            dates[index++] = ch;
        ch = fgetc(r_dates);
  }
  while( dates[index] == 32 )
         index--;
  index++;
  dates[index]='\0';
  printf("dates : %s\n",dates);

  for (index=0;index<3;index++){
       convert[index]=dates[index];
  }
  convert[index]='\0';
  printf("convert : %s\n", convert);
  mo = convert_to_integer(convert);
  printf("mo : %2d\n", mo);
  if (mo == 100)
     printf("can't convert dates to integers\n");

  sscanf(dates, "%3c/%2d/%2d %2d:%2d", month, &dy, &yr, &hr, &mn);
  sprintf(cur_dates, "20%02d-%02d-%02d %02d:%02d", yr, mo, dy, hr, mn);
  printf("cur_dates : %s\n",cur_dates);

  /*********************************************************/
  /************* REARRANGE THE POLL.TXT ********************/
  /*********************************************************/

  system("./delete_bracket /usr/local/lsd/src/autoqc/poll.txt > polling");
  /* The file "polling" have column forms separated by spaces */

  system("cut -d' ' -f2 polling > tmp0");
  system("cut -d' ' -f3- polling > tmp1");
  system("cut -d' ' -f1 polling | paste tmp0 - tmp1 > polling2");
  system("sort -u polling2 > polling");
  system("cut -f1 polling > polling2");
  /* the file "polling2" includes only site numbers */
  /* the file "polling" includes site numbers and serial numbers */

  /*********************************************************/
  /********** REDUCE THE OUTPUT OF THE CREATE_LOG.C ***********/
  /*********************************************************/
```

```
  if ((r_buffer = fopen ("./created_log","r")) == NULL)
       printf("The file not opened\n");

  num_fragments = Calculate_Fragments("created_log");
  printf("num_fragments: %d\n",num_fragments);

  for (i = 0; i < num_fragments; i++) {
/*-------------------------------------------------------------------*/
/*----'do loop' and 'for loop' are implementations of the "itoa()"----*/
/*-As these're only related to name files, you may ignore them parts--*/
/*-------------------------------------------------------------------*/
       n = i;
       j = 0;
       do {
           num[j++] = n % 10 + '0';
       } while ((n /= 10) > 0);
       num[j] = '\0';
       for (j = 0, k = strlen(num)-1; j < k; j++, k--) {
            c = num[j];
            num[j] = num[k];
            num[k] = c;
       }

/*- The "mid[i]" includes each fragment name of "pollc180YYJJJ.log" -*/
/*- Each fragment name length is restricted by MAX_CHAR characters  -*/
       if ((mid[i] = (char *)malloc(MAX_CHAR)) == NULL) {
                    printf("Not enough space\n");
                    exit(-1);
       }

/*--- Each fragment is named "mid0","mid1","mid2",... ,and so on ---*/
       strcpy(mid[i], "mid");
       strcat(mid[i], num);

/*-------------------------------------------------------------------*/
/*---------- Until this point, just have a disregard ----------------*/
/*-------------------------------------------------------------------*/

/*-- Func Split_into_Fragments(,) puts fragments of "created_log" ---*/
/*-- into files called "mid0","mid1","mid2",...,and so on.       ----*/

       if ((w_buffer = fopen(mid[i],"w")) == NULL)
            printf(" The file not existed\n");
            Split_into_Fragments(r_buffer, w_buffer);
            fclose(w_buffer);

            sprintf(command, "./filter_str %s > filtered_file",mid[i]);
            system(command);
            system("cat filtered_file >> target_file");

/* All other files except for the final "target_file" will be removed*/
            sprintf(command, "yes|rm %s", mid[i]);
            system(command);

  } /*------- for loop executed by num_files times ------------*/
  fclose(r_buffer);
  system("yes|rm created_log");

  system("./filter_str target_file > info");
  system("rm filtered_file target_file");

  /*************************************************************/
  /****************** UNION PROCESS ****************************/
  /*************************************************************/

  system("sort -u info > input.log");
  /* the "input.log" includes unique sorted site information failed*/

  system("fgrep -f polling2  input.log > output");
  system("join polling output > out.log");

  /*************************************************************/
  /****************** NAME THE OUTPUT **************************/
  /*************************************************************/

  if ((name = (char *)malloc(MAX_LINE)) == NULL) {
       printf("Not enough space\n");
       exit(-1);
  }
  strcpy(name, "/usr/local/lsd/src/autoqc/faillist.");
```

```
  if ( (read_log = fopen("/usr/local/lsd/src/autoqc/poll.txt","r")) == NULL )
        printf("The file not opened\n");

  index=0;
  ch=fgetc(read_log);
  while(ch == 10)
        ch = fgetc(read_log);

  while(ch != 10) {
        if (ch == 47)/* ASCII code 47 is matched with "/" */
            index--;
        else if (ch == 32)/* ASCII code 32 is matched with the space key*/
            temp[index]= 95;/* ASCII code 95 is matched with "_" */
        else
            temp[index] = ch;
        ch = fgetc(read_log);
        index++;
  }
  fclose(read_log);
  while( temp[index] == 95 )
         index--;
  index++;
  temp[index]='\0';

  for (i=0;i<7;i++){
       date[i]=temp[i];
  }
  date[i]='\0';

  strcat(name, date);
  /* this is the output file name */

  /*****************************************************/
  /************** FIANL FILTERING **********************/
  /*****************************************************/

  sprintf(command, "./filter_str out.log > %s",name);
  system(command);/* the file "faillist.******" is the result merged */

  system("rm tmp0 tmp1 polling polling2 info input.log output out.log");
  /* All interim output files except for a final result are erased */

  /*****************************************************/
  /************** INSERT INTO FAILLIST TABLE ***********/
  /*****************************************************/

  faillist_load(name, cur_dates);

  /*****************************************************/
  /***********  RENAME THE OUTPUT FILE *****************/
  /*****************************************************/
      {
         char newname[100];
         strcpy(newname,name);
         strcat(newname,"_loaded");
         sprintf(command, "mv %s %s ",name, newname);
         system(command);
      }

  return 0;

}

int convert_to_integer(char * convert)
{

    if (strcmp(convert, "Jan")==0)
        return (1);
    else if (strcmp(convert, "Feb")==0)
        return (2);
    else if (strcmp(convert, "Mar")==0)
        return (3);
    else if (strcmp(convert, "Apr")==0)
        return (4);
    else if (strcmp(convert, "May")==0)
        return (5);
    else if (strcmp(convert, "Jun")==0)
        return (6);
    else if (strcmp(convert, "Jul")==0)
        return (7);
```

```
    else if (strcmp(convert, "Aug")==0)
        return (8);
    else if (strcmp(convert, "Sep")==0)
        return (9);
    else if (strcmp(convert, "Oct")==0)
        return (10);
    else if (strcmp(convert, "Nov")==0)
        return (11);
    else if (strcmp(convert, "Dec")==0)
        return (12);
    else
        return (100);
}

int Calculate_Fragments(char buffer[MAX_LINE])
{
    int org_file_size, num_fragments;
    char command[MAX_LINE];

    /* Count total number of characters on the "pollc180YYJJJ.log"*/
    /* The number counted is written in the file "tmp"            */
    sprintf(command, "wc -c %s > tmp",buffer);
    system(command);

    /* Then the total number's stored in the variable "org_file_size" */
    if ((tmp_ptr = fopen("tmp", "r")) == NULL)
         printf("The file not opened\n");
    fscanf(tmp_ptr, "%d", &org_file_size);
    fclose(tmp_ptr);
    system("yes|rm tmp");

/* The input file "created_log" will be separated by 26000 characters.*/ /* The number of
fragments of the "created_log" is stored in the */
/* variable "num_files". How many times 'for loop' must be repeated */
/* is determined by the number "num_files" */

    num_fragments = (int)ceil((org_file_size-1) / 500000.00);

    return (num_fragments);
}

void Split_into_Fragments(FILE *r_buffer, FILE *w_buffer)
{
     char buffer[500000] = {0};

     fread( buffer, sizeof(char), 500000 , r_buffer);
     fprintf(w_buffer,"%s\n", buffer);
}

void faillist_load(char *rawname, char *cur_dates)
{
     FILE *rawfile;
     int ch,i,index;
     int site_number, serial_number;
     char  n_site[5], n_serial[5];
     char  baud_rate[15], phone_number[30],error_description[1000];

     if ((rawfile = fopen(rawname,"r")) == NULL)
     {
          fprintf (stderr,"There is no rawfile -- %s\n", rawname);
          exit(-1);
     }

     ch = fgetc(rawfile);
     while(ch == 10)
           ch = fgetc(rawfile);

           while(ch != EOF){

/* if the first character in the new line,new_ch,corresponds to */
/* '0' <= ch <= '9', this is the beginning of the logging information */
/* Sometimes a line begins with a character '-' */

                while(ch >= 48 && ch <= 57 || ch == 45){
                      i=0;
                      while(ch !=32) {
                            n_site[i++]=ch;
                            ch = fgetc(rawfile);
                      }
                      n_site[i] = '\0';
                      site_number = atoi(n_site);
```

```
                        while(ch == 32)
                              ch = fgetc(rawfile);

                        i=0;
                        while(ch !=32) {
                              n_serial[i++]=ch;
                              ch = fgetc(rawfile);
                        }
                        n_serial[i] = '\0';
                        serial_number = atoi(n_serial);

                        while(ch == 32)
                              ch = fgetc(rawfile);

                        i=0;
                        while(ch != 41) {
                              baud_rate[i++]=ch;
                              ch = fgetc(rawfile);
                        }
                        baud_rate[i++] = ')';
                        baud_rate[i] = '\0';

                        ch = fgetc(rawfile);
                        while(ch == 32)
                              ch = fgetc(rawfile);
                        i=0;
                        while(ch !=32) {
                              phone_number[i++]=ch;
                              ch = fgetc(rawfile);
                        }
                        phone_number[i] = '\0';

                        while(ch == 32)
                              ch = fgetc(rawfile);

                        i=0;
                        while(ch !=10) {
                              error_description[i++]=ch;
                              ch = fgetc(rawfile);
                        }
                        error_description[i] = '\0';

insert_rows(cur_dates,site_number,serial_number,baud_rate,phone_number,error_description);

                        ch = fgetc(rawfile);
                      }/* One Line read */
          } /* The End Of File comes up */
          fclose(rawfile);
          fclose(w_stream);
}

void insert_rows(dates,site,serial,baud,phone,error)
$ parameter short site;
$ parameter short serial;
$ char *baud;
$ char *phone;
$ char *error;
$ char *dates;
{
  $ database lsd@lsol;

  $ insert into lsd@lsol:faillist
    (date, site_number, serial_number, baud_rate, phone_number, error_description)
    values ($dates, $site, $serial, $baud, $phone, $error);

  $ close database;
}
```

6.2 Automatic Data Quality Process-Tier 2 (Source)

autoqc2.ec

```
||============================================================
||  usage : to catch the following data from each raw file
||          - values flagged(F/U/O/N)
||          - values out of range
||          - jumpings betweeen TSRs/timestamps
||          - zero values
||          and to store these data into appropriate tables
||  tables: - power_fail_log
||          - out_of_range
||          - missing_records
||          - zeros
||  loading the data on power_fail_log and out_of_range is executed
||  by reload process. autoqc2 catches these data loaded by reload.
||  input : /prod/raw/sssyyjjj.raw_loaded (ex.17801002.raw_loaded)
||  output: /usr/local/lsd/src/autoqc/sssyyjjj_autoqc2
||          all loaded on above four tables is written on this file.
||  Copyright  2001-2002,
||  Texas Engineering Experiment Station,
||  Texas A&M University.
||  Architect: Jim Sweeney, Programmer: YoonCheung Chang
||============================================================

#include<string.h>
#include<stdio.h>
#include<stdlib.h>
$include sqlca;
$include sqltypes;

#define MAX_TIME 20
#define MAX_LINE 500
#define MAX_DATA 1000
#define MAX_CH   100

#define autoqc2_path "/usr/local/lsd/src/autoqc/"

void missing_records(int, int,char time[MAX_LINE][MAX_TIME]);
void check_zeros(int, int, char time[MAX_LINE][MAX_TIME], char data[MAX_LINE][MAX_DATA]);

FILE *r_input, *w_missing_records, *w_zeros;

main(int argc,char **argv)
{

 char data[MAX_LINE][MAX_DATA], time[MAX_LINE][MAX_TIME];
 char input_fullname[100],input_basename[20];
 char *line_ptr;
 int i,j,k,index, ch, num_lines;
 int logger_id;
 long juldate;

 /*  check arguments */
 if (argc!=2) { printf("Usage: autoqc2 input_fullname\n"); exit(-1);}

 /* To get the input file */
 strcpy(input_fullname,argv[1]); /*strcpy(input_name,"17801002.raw_loaded");*/
 printf("input_fullname: %s\n",input_fullname);

 /* To get the logger_id & julian date*/
 line_ptr = strpbrk(input_fullname, "\.");
 line_ptr = line_ptr-8;
 printf("line_ptr:%c\n",*line_ptr);
 i=0;
 while(*line_ptr != '\0'){
      input_basename[i++]=*line_ptr;
      line_ptr++;
 }
 input_basename[i]='\0';
 printf("input_basename: %s\n",input_basename);
 sscanf(input_basename,"%03d%05ld.raw_loaded",&logger_id,&juldate);

 printf("logger_id : %d\n", logger_id);
 printf("juldate : %05d\n", juldate);
```

```
/*******************************************************/
/*** timestamps, and channel data are put into *****/
/* appropriate arrays, time and data, respectively */
/*******************************************************/

if ( (r_input = fopen (input_fullname,"r")) == NULL)
      printf("The file not opened\n");

ch = fgetc(r_input);
while(ch == 10)
      ch = fgetc(r_input);
/*printf("the beginning of the file: %d\n", ch);*/

index = 0;
while(ch != EOF) {
      while(ch != 34)
            ch = fgetc(r_input);

     /* For storing the data values */
      while(ch > 57 || ch < 48)
            ch = fgetc(r_input);

      i = 0;
      while(ch != 10) {
            data[index][i++] = ch;
            ch = fgetc(r_input);
      }
      data[index][i++] = 32;
      data[index][i] ='\0';

      /*printf("data[%d]: %s\n", index, data[index]);*/

      ch = fgetc(r_input);
      index++;

}/* The end of file comes up */

num_lines = index;
printf("the number of lines: %d\n", num_lines);

fseek(r_input, 0L, SEEK_SET);
ch = fgetc(r_input);

while(ch == 10)
      ch = fgetc(r_input);
/*printf("the beginning of the file: %d\n", ch);*/

index = 0;
while(ch != EOF) {
      for(i=0;i<18;i++){
          time[index][i] = ch;
          ch = fgetc(r_input);
      }
      time[index][i]='\0';
     /* printf("time[%d]: %s\n", index, time[index]);*/
      while(ch != 10)
            ch = fgetc(r_input);
      ch = fgetc(r_input);
      index++;
}/* The end of file comes up */

missing_records(logger_id, num_lines, time);
check_zeros(logger_id, num_lines, time, data);

/* create the output file */
{
  char output_name[100],command1[200],command2[200];

  sprintf(output_name, "%s%03d%05d_autoqc2",autoqc2_path,logger_id,juldate);
  sprintf(command1, "cat %sout_of_range.log %spower_fail_log.log %smissing_records.log
%szeros.log > %s",autoqc2_path,autoqc2_path,autoqc2_path,autoqc2_path,output_name);
  system(command1);
  sprintf(command2,"rm %sout_of_range.log %spower_fail_log.log %smissing_records.log
%szeros.log",autoqc2_path,autoqc2_path,autoqc2_path,autoqc2_path);
  system(command2);
 }
}

void missing_records(logger_id,num_lines,time)
$parameter int logger_id;
int num_lines;
```

```
char time[MAX_LINE][MAX_TIME];
{
 int i, j, k, mon_index, jumping_buff[100], num_time_jumping;
 int time_index, begin_hour, end_hour, begin_min, end_min;
 int  *mo,*dy,*yr,*hour,*min,*sec,diff_hour;
 int be_mo, be_dy, be_yr, en_mo, en_dy, en_yr;
 char end_dates[MAX_TIME];

 /* int months[12] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};*/
 int num_days[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

 $ int ch_id;
 $ int raw_cp;
 $ int int_period;
 $ int count;
 $ char count_query[100];
 $ char begin_dates[20];

 if ((w_missing_records = fopen("/usr/local/lsd/src/autoqc/missing_records.log","w")) == NULL)
      printf("The file not existed\n");

 fprintf(w_missing_records,"\nmissing_records\(skipping TSRs/timestamps\)\n\n");
 fprintf(w_missing_records,"ch_id   timestamp\n\n");

 $ database lsd@lsol;

 $ select int_period into $int_period
   from lsd@lsol:chids
   where ch_id in
   (select min(ch_id)
    from lsd@lsol:rawmap
    where logger_id=$logger_id);

 printf("int_period: %d\n",int_period);

 mo = (int *)malloc((size_t)(num_lines*sizeof(int)));
 dy = (int *)malloc((size_t)(num_lines*sizeof(int)));
 yr = (int *)malloc((size_t)(num_lines*sizeof(int)));
 hour =(int *)malloc((size_t)(num_lines*sizeof(int)));
 min = (int *)malloc((size_t)(num_lines*sizeof(int)));
 sec = (int *)malloc((size_t)(num_lines*sizeof(int)));

 for(i=0;i<num_lines;i++){
     sscanf(time[i],"%2d/%2d/%2d  %2d:%2d:%2d",mo+i,dy+i,yr+i,hour+i,min+i,sec+i);
     /*printf("mo:%d dy:%d yr:%d hour:%d min:%d\n",
*(mo+i),*(dy+i),*(yr+i),*(hour+i),*(min+i));*/
 }

 /* calculate the initial time and check the jumping */
 if (int_period > 60){
     printf("int_period=144\n");
 }
 else{
      if(int_period == 60 || int_period == 30){
         j=0;
         for(i=1;i<num_lines;i++){
             diff_hour=*(hour+i)-*(hour+(i-1));
             /*printf("diff_hour:%d\n",diff_hour);*/
             if(diff_hour != 1 && diff_hour != 0 && diff_hour != -23)
                jumping_buff[j++]=i;
         }
         jumping_buff[j]='\0';
         num_time_jumping=j;
         printf("the number of jumpings: %d\n", num_time_jumping);
      }
      else if(int_period == 15){
             printf("int_period=15\n");
      }
      else {
             printf("another int_period\n");
      }
 }

 if (num_time_jumping > 0){

     for(i=0;i<num_time_jumping;i++){
        time_index=jumping_buff[i];

        if (int_period > 60){
            be_dy = *(dy+(time_index-1))+1;
            be_mo = *(mo+(time_index-1));
```

```
            be_yr = *(yr+(time_index-1));

            mon_index = be_mo-1;
            if(be_dy == num_days[mon_index]+1){
               be_dy = 1;
               be_mo = be_mo+1;
               if(be_mo == 13){
                  be_mo = 1;
                  be_yr = be_yr + 1;
               }
            }

            sprintf(begin_dates,"20%02d-%02d-%02d",be_yr,be_mo,be_dy);
            sprintf(end_dates, "20%02d-%02d-
%02d",*(yr+time_index),*(mo+time_index),*(dy+time_index));

            printf("begin_dates: %s\n",begin_dates);
            printf("end_dates: %s\n",end_dates);
        }

        else {
             if(int_period == 60 || int_period == 30){

                begin_hour=*(hour+(time_index-1))+1;
                be_dy = *(dy+(time_index-1));
                be_mo = *(mo+(time_index-1));
                be_yr = *(yr+(time_index-1));

                if (begin_hour == 24){
                    begin_hour = 0;
                    be_dy = be_dy + 1;

                    mon_index = be_mo-1;
                    if(be_dy == num_days[mon_index]+1){
                       be_dy = 1;
                       be_mo = be_mo+1;
                       if(be_mo == 13){
                          be_mo = 1;
                          be_yr = be_yr + 1;
                       }
                    }
                }

                sprintf(begin_dates,"20%02d-%02d-%02d %02d",be_yr,be_mo,be_dy,begin_hour);
                sprintf(end_dates, "20%02d-%02d-%02d
%02d",*(yr+time_index),*(mo+time_index),*(dy+time_index),*(hour+time_index));

                printf("begin_dates : %s\n",begin_dates);
                printf("end_dates : %s\n",end_dates);
            }
            else if(int_period == 15) {

                begin_min = *(min+(time_index-1)) + int_period;
                begin_hour = *(hour+(time_index-1));
                be_dy = *(dy+(time_index-1));
                be_mo = *(mo+(time_index-1));
                be_yr = *(yr+(time_index-1));

                if (begin_min == 60){
                    begin_min=0;
                    begin_hour=begin_hour+1;
                    if (begin_hour == 24){
                        begin_hour = 0;
                        be_dy = be_dy + 1;

                        mon_index = be_mo-1;
                        if(be_dy == num_days[mon_index]+1){
                           be_dy = 1;
                           be_mo = be_mo+1;
                           if(be_mo == 13){
                              be_mo = 1;
                              be_yr = be_yr + 1;
                           }
                        }
                    }
                }

                sprintf(begin_dates,"20%02d-%02d-%02d
%02d:%02d",be_yr,be_mo,be_dy,begin_hour,begin_min);
```

```
                sprintf(end_dates, "20%02d-%02d-%02d
%02d:%02d",*(yr+time_index),*(mo+time_index),*(dy+time_index),*(hour+time_index),*(min+time_index
));

                printf("begin_dates : %s\n",begin_dates);
                printf("end_dates : %s\n",end_dates);

            }/* int_period = 15 min*/
            else {
                printf("There is the other int_period\n");
            }

        }/* int_period = 60, 30, and 15 */

        $ declare chid_curs cursor for
            select ch_id, raw_cp
              from lsd@lsol:rawmap
                where logger_id = $logger_id
                order by raw_cp;

        while(strcmp(begin_dates,end_dates) != 0){

            $ open chid_curs;

            $ fetch chid_curs into $ch_id, $raw_cp;

            if (sqlca.sqlcode == SQLNOTFOUND)
            {
                printf("No ch_ids for this logger - %d \n", logger_id);
                exit(-2);
            }

            while (sqlca.sqlcode != SQLNOTFOUND)
            {
              /*printf("begin_dates: %s ch_id: %d raw_cp: %d\n",begin_dates,ch_id,raw_cp);*/

              sprintf(count_query,
              "select count(*) from lsd@lsol:ch%04d"
              " where timestamp=\"%s\"",ch_id,begin_dates);

              $prepare count_stmt from $count_query;

              $declare count_curs cursor for count_stmt;

              $open count_curs;

              $fetch count_curs into $count;

              /* printf("count: %d\n",count);*/

              $close count_curs;

              if(count == 0)
              {
                printf("need to insert this timestamp into the missing_records table\n");
                $ insert into
                    lsd@lsol:missing_records(ch_id, timestamp)
                    values($ch_id, $begin_dates);

                fprintf(w_missing_records,"%d   %s\n",ch_id,begin_dates);
              }
              else{
                printf("Not need to insert this timestamp\n");
              }
              $ fetch chid_curs into $ch_id, $raw_cp;
            }/* while */

            $ close chid_curs;

            if (int_period > 60){
                be_dy = be_dy+1;

                mon_index = be_mo-1;
                if(be_dy == num_days[mon_index]+1){
                   be_dy = 1;
                   be_mo = be_mo+1;
                   if(be_mo == 13){
                      be_mo = 1;
                      be_yr = be_yr + 1;
                   }
                }
```

```
                    sprintf(begin_dates,"20%02d-%02d-%02d",be_yr,be_mo,be_dy);
                    printf("begin_dates: %s\n",begin_dates);
                }

                else {
                      if(int_period == 60 || int_period == 30){

                         begin_hour = begin_hour+1;

                         if (begin_hour == 24){
                             begin_hour = 0;
                             be_dy = be_dy + 1;

                             mon_index = be_mo-1;
                             if(be_dy == num_days[mon_index]+1){
                                be_dy = 1;
                                be_mo = be_mo+1;
                                if(be_mo == 13){
                                   be_mo = 1;
                                   be_yr = be_yr + 1;
                                }
                             }
                         }

                         sprintf(begin_dates,"20%02d-%02d-%02d %02d",be_yr,be_mo,be_dy,begin_hour);
                         printf("begin_dates : %s\n",begin_dates);
                      }
                      else if(int_period == 15) {

                           begin_min = begin_min + int_period;

                           if (begin_min == 60){
                               begin_min=0;
                               begin_hour=begin_hour+1;
                               if (begin_hour == 24){
                                   begin_hour = 0;
                                   be_dy = be_dy + 1;

                                   mon_index = be_mo-1;
                                   if(be_dy == num_days[mon_index]+1){
                                      be_dy = 1;
                                      be_mo = be_mo+1;
                                      if(be_mo == 13){
                                         be_mo = 1;
                                         be_yr = be_yr + 1;
                                      }
                                   }
                               }
                           }

                           sprintf(begin_dates,"20%02d-%02d-%02d
%02d:%02d",be_yr,be_mo,be_dy,begin_hour,begin_min);
                           printf("begin_dates : %s\n",begin_dates);
                      }/* int_period = 15 min*/
                      else
                          printf("There is the other int_period\n");
                }/* int_period = 60, 30, and 15 */
          }/* until the begin_dates is equal to the end_dates */
      }/* for statement repeated num_time_jumping times */
  }/* if time_jumping exists */
  else
      printf("There is no jumping of time\n");

  fclose(w_missing_records);
  $ close database;

}

void check_zeros(logger_id,num_lines,time,data)
$parameter int logger_id;
int num_lines;
char time[MAX_LINE][MAX_TIME];
char data[MAX_LINE][MAX_DATA];
{
  int i, k, j, index;
  char temp[10];
  int value[MAX_CH];

  $ int ch_id;
  $ int raw_cp;
```

```
  $ char timestamp[20];

  if ((w_zeros = fopen("/usr/local/lsd/src/autoqc/zeros.log","w")) == NULL)
       printf(" The file not existed\n");

  fprintf(w_zeros,"\nzero values\n\n");
  fprintf(w_zeros,"ch_id   timestamp\n\n");

  $ database lsd@lsol;

  for (index=0;index<num_lines;index++){
       i=0; k=0;
       while(data[index][i] != NULL) {
             j = 0;
             while(data[index][i] != 32){
                   temp[j] = data[index][i];
                   j++; i++;
             }
             temp[j]='\0';

             value[k]=atoi(temp);
             /*printf("value[%d]: %d\n", k, value[k]);*/
             if (value[k] == 0){
                 raw_cp = k+1;
                 strcpy(timestamp,time[index]);
                 /*printf("timestamp:%s\n", timestamp);*/

                 $ select ch_id into $ch_id from lsd@lsol:rawmap
                   where logger_id = $logger_id and raw_cp = $raw_cp;

                 /*printf("ch_id: %d\n", ch_id);*/

                 $ insert into
                   lsd@lsol:zeros(ch_id, datatime_stamp)
                    values($ch_id, $timestamp);

                   fprintf(w_zeros,"%d   %s\n",ch_id,timestamp);
             }
             k++;
             while(data[index][i] == 32)
                    i++;
        }
  }

  fclose(w_zeros);
  $ close database;
}
```